Exercise 1: Tuning k-NN

In this exercise we will perform hyperparameter optimization (HPO) for the task of classifying the `credit risk` data with a k-NN classifier.

`R` exercise:
The `kknn` implementation used by `mlr3` contains several hyperparameters, three of which are to be tuned for our prediction:

- k (number of neighbors)
- `kernel`
- `scale`

a) Describe briefly the role of each hyperparameter in the learning algorithm – which effects can be expected by altering them?

b) In `mlr3` (using the `mlr3tuning` library), define an appropriate search space to tune over. We want to explore a range between 1 and 100 for k and the kernel to be chosen from "rectangular", "epanechnikov", "gaussian", "optimal".

c) Perform the tuning procedure using `TuningInstanceSingleCrit`. Set aside 200 test observations first. Use 5-fold cross validation and random search, and terminate the process after either 30 seconds or 200 evaluations.

d) You realize that a high AUC is the performance measure you are actually interested in. Modify the HPO procedure such that performance is optimized w.r.t. AUC.

e) Visualize the tuning result with a suitable command. What do you observe regarding the impact of different hyperparameters on predictive performance? What are limits of such a form of analysis?

f) After analyzing the tuning results, you notice that changes in k are more influential for smaller neighborhoods. Re-run the HPO procedure with a log-transformation for k.

g) With the hyperparameter configuration found via HPO, fit the model on all training observations and compute the AUC on your test data.

`Python` exercise:
The `KNeighborsClassifier` implementation used by `sklearn` contains several hyperparameters, three of which are to be tuned for our prediction:

- `n_neighbors`
- `weigths`
- `metric`

a) Describe briefly the role of each hyperparameter in the learning algorithm – which effects can be expected by altering them? Furthermore, read the credit_for_py.csv, separate 138 test observations and perform necessary preprocessing steps.
(Hint: Apply a `StandardScaler` on your feature space. What effect does scaling have on your k-NN-model?)

b) Define an appropriate search space to tune over. We want to explore a range between 1 and 100 for `n_neighbors` and the distance calculation to be chosen from "uniform", "manhattan", "euclidean", "cosine".

c) Perform the tuning procedure using `RandomizedSearchCV`. Use 5-fold cross validation, and terminate the process after 200 iterations. Also, utilize parallelization to fasten the computation.

d) You realize that a high AUC is the performance measure you are actually interested in. Modify the HPO procedure such that performance is optimized w.r.t. AUC.

e) You are interested in possible under- and overfitting of your hyperparameter setting. Use `validation_curve` from `sklearn.model_selection` to retrieve the training scores and cross-validation scores for a 5-fold-CV, depending on the AUC metric.
Visualize the tuning result with a suitable command. You may use the function provided below or a self-defined function.
Re-run the evaluation with unscaled features.

   What do you observe regarding the impact of different hyperparameters and scaling on predictive performance? What are limits of such a form of analysis?

f) After analyzing the tuning results, you notice that changes in `n_neighbors` are more influential for smaller neighborhoods. Re-run the HPO procedure with a log-transformation for `n_neighbors` parameter list.

g) With the hyperparameter configuration found via HPO, fit the model on all training observations and compute the AUC on your test data. Could you see any effect of the log-transformation for `n_neighbors`?

Function for plotting a validation curve:

```
def plot_validation(train_scores, test_scores, param, param_range):
    """
    Plot the validation curve for a given hyperparameter.

    Parameters:
    ----------
    train_scores : array-like of shape (n_param_values, n_folds)
        Training scores for each hyperparameter value and fold.
    test_scores : array-like of shape (n_param_values, n_folds)
        Test scores (e.g., cross-validation scores) for each hyperparameter value and fold.
    param : str
        Name of the hyperparameter being varied.
    param_range : array-like of shape (n_param_values,)
        Range of values for the hyperparameter.

    Returns:
    -------
    None
        The function plots the validation curve.

    """
    train_scores_mean = np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    test_scores_std = np.std(test_scores, axis=1)

    plt.title("Validation Curve with KNN")
    plt.xlabel(param)
    plt.ylabel("Score")
    plt.ylim(0.0, 1.1)
    lw = 2
    plt.plot(
        param_range, train_scores_mean, label="Training score", color="darkorange", lw=lw
    )
    plt.fill_between(
        param_range,
        train_scores_mean - train_scores_std,
        train_scores_mean + train_scores_std,
        alpha=0.2,
        color="darkorange",
        lw=lw,
    )
    plt.plot(
        param_range, test_scores_mean, label="Cross-validation score", color="navy", lw=lw
    )
    plt.fill_between(
        param_range,
        test_scores_mean - test_scores_std,
        test_scores_mean + test_scores_std,
        alpha=0.2,
        color="navy",
        lw=lw,
    )
    plt.legend(loc="best")
    plt.show()
```